Санкт-Петербургский Государственный Университет
Прикладной математики - Процессов Управления

Кафедра Технологии Программирования

Аксенова Мария Владимировна

Автодополнение кода с использованием нейросетевых языковых моделей

Бакалаврская работа

Научный руководитель:
ст. преп. Мишенин А. Н.

Санкт-Петербург
2021

Оглавление

Введение

Мир и человечество развиваются и не стоят на месте, однако с течением времени растет не только прогресс, но и количество данных. Исследования любой из сфер нашей жизни требуют новых данных, и многие из них хранятся в текстовом формате. В связи с этим появляется большое количество методов обработки и генерирования текстовых данных: например перевод из одного языка в другой, генерирование текста, анализ текста и его распознавание, и многие другие задачи. Для использования любого из этих методов необходимо написать свою соответствующую программу, которая тоже является текстовыми данными.

Современные методы обработки естественного языка обеспечивают нам решение множества задач, связанных с текстом, но несмотря на это многие из таких методов не применялись ранее к техническим языкам, таким как языки программирования

Любая существующая программа написана на особенном языке программирования, который имеет свой собственный синтаксис. Каждый из таких языков можно трактовать как новый иностранный язык. С ними проводятся различные операции: это не только перевод и обработка языка, но и самостоятельное генерирование предложений, текстов, прогнозирование фраз по их началу. Похожий подход возможно использовать для того , чтобы облегчить написание кода и сократить время разработки продукта. Такой подход, позволяющий ускорить написание программы, называется автодополнением.

Проблема скорости написания кода часто встречается и остро стоит у профессионалов в данной области. Это связано с тем, что им необходимо писать похожие части кода из раза в раз. Для специалистов было бы полезно, если бы программа умела предугадывать и подсказывать наиболее подходящие слова, которые программист теоретически мог бы хотеть написать.

Автоматическое дополнение кода – технология,обеспечивающая интерактивный ввод кода, предсказывая его по уже введенной части.Она необходима в первую очередь для того, чтобы ускорить написание каких-

либо шаблонных и наиболее часто употребляемых частей кода.

В данной дипломной работе будет рассмотрена проблема скорости написания кода.Способы ее решения будут заключаться в поиске специальных моделей, различных методов, уместности их применения, способах их улучшения, и сравнении их плюсов и недостатков.

Воплотить это в жизнь можно при помощи машинного обучения. Написание автодополнения — это трудоемкая задача. Каждый язык программирования имеет свою структуру и свои особенности, поэтому нужны особые наборы данных. В настоящей работе был выбран язык Python, который является одним из самых наиболее часто используемых языков, так как позволяет решать различные задачи : начиная от разработки программной части , заканчивая проектами по машинному обучению. Обучив технологию на одном языке, ее можно будет распространить на другой, заменив обучающее множество данных.

Решить задачу можно различными способами. Один из первых методов — генерирование последовательностей при помощи seq2seq модели,чья архитектура реализована при помощи рекуррентных нейросетей глубинного обучения: на такой их разновидности как LSTM(long-short term memory). С развитием технологий появилась state-of-the-art модель BERT, соединяющая в себе идею маскирования токенов и архитектуру transformer. Она сделала прорыв в обработке естественного языка и показала важность авторегрессионных моделей, совмещающих прямой и обратный ход и учитывание контекста предложения.

Для практического применения в данной работе в основе будет рассмотрена модель BERT вместе со своими достоинствами и недостатками. Основой является идея маскирования токенов и их предсказывания по контексту, однако она не решает проблему влияния замаскированных токенов друг на друга. В связи с этим будет рассмотрена модель BISon, которая должна решить эту проблему, но эта модель применялась только к корпусу текста, не включая корпус кода. В работе будет рассмотрено применение данных архитектур к задаче автодополнения кода.

Актуальность работы

Актуальность работы обусловлена практическим аспектом поднятой проблемы. Сфера it-технологий с развитием мира и машинного обучения находится в стадии постоянного совершенствования. Практически каждый it-продукт связан с программированием и написанием кода соответственно. Ускорение написания кода поможет программистам тратить меньшее количество времени на создание продукта.

Автодополнение также помогает начинающим специалистам быстрее выучить язык, а опытным профессионалам ускорить изучение новых библиотек. Это связано с тем, что технология предлагает различные варианты дополнения одной и той же строки. Просматривая их, человек видит и запоминает разнообразие методов, предлагаемых языком. Открывая неизвестные ему методы, узнает новую информацию и улучшает свою квалификацию.

Цель и задачи работы

Цель работы : исследование различных нейросетевых методов для задачи автодополнения кода и генерирование новых методов на основе изученных.

Задачи работы:

- Изучить научную литературу по теме исследования
- Рассмотреть популярные методы, используемые для генерирования текста
- Рассмотреть применение существующих методов к задаче автодополнения кода
- Сформулировать собственные идеи, которые можно использовать для автодополнения кода
- Реализовать наиболее возможные в плане обучения идеи

Постановка задачи

Рассматриваться будут авторегрессионные модели: модели, функция которых зависит только от заранее известных элементов.

Пусть имеется конечное множество элементов размерности n и его разбиение на два непересекающихся подмножества.

$x = [x_1, x_2, ..., x_n]$

$x' \cup x'' \in x,\ x' \cap x'' = \varnothing$

Необходимо узнать условную вероятность появления токенов x' в данной последовательности при известных x".

$Prob(x'/x'')-?$

Это является стандартной задачей языкового моделирования. Решить ее может функция, принимающая на вход несколько токенов и возвращающая распределение вероятности появления каждого токена, входящего в состав словаря.

$f(x') = Prob(x_i/x_1...x_{i-1}),\ i = 1...n, x' \in x.\ x'$

В данной работе мы будем рассматривать в качестве решения элементы $y_i \in x$, имеющие наибольшую вероятность появления при заданных условиях, поэтому формула примет вид:

$y_i = argmax(Prob(x_i/x_1...x_{i-1})),\ i = 1...n, x' \in x.\ x'$

Дополнением может быть также не один элемент, а их совокупность, поэтому конечную формулу можно записать в следующем виде‘:

$y_{j_i}...y_{j_k} = argmax(Prob(x_{j_i}...x_{j_k}/x_1...x_{i-1})),\ i < k < n$

Обзор литературы

В приложенной литературе рассматриваются различные исследования рекуррентных архитектур. Приложение [1] рассказывает нам о наиболее известных первых рекурретных архитектурах.

Далее происходит переход к более сложным архитектурам, поэтому в приложении[8] находится статья с описанием seq2seq модели, которая является базовой для понимания статей[9],[2]. В приложении[9] авторы повествуют о стандартной Transformer модели, в дополнение к [9] присутствует более иллюстрированное приложение[2], которое поможет читателю более уверенно разобраться в данной архитектуре.

В приложении[4] происходит переход к идеям улучшенной Transformer архитектуры BERT.

Завершают приложения [6] и [3], которые являются ключевыми для понимания данной работы. Они рассказывают нам об архитектурах code2seq и BISon, суммируют и подводят итог проделанному исследованию литературы, дают нам возможность представления собственных идей.

1. Обзор существующих моделей

В данном разделе мы рассмотрим особенности некоторых моделей, позволяющих решить поставленную задачу. Обзор будет производиться в порядке усложнения архитектур и их исторического появления. Сначала будут рассмотрены классические рекуррентные модели, порождающие encoder-decoder архитектуры, далее происходит переход к современным state-of-the-art моделям и исследованию их недостатков.

1.1. Рекуррентные архитектуры

В настоящее время уже существуют успешные способы генерировать текст. Началось все с появлением рекуррентных нейронных сетей (RNN). Это вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий, в которых важно не только содержание, но и порядок. В отличие от известных многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины.

Таким образом состояние y_t в момент времени t с входной последовательностью x_t можно выразить следующей формулой:

$y_t = f(y_t - 1, x_t)$

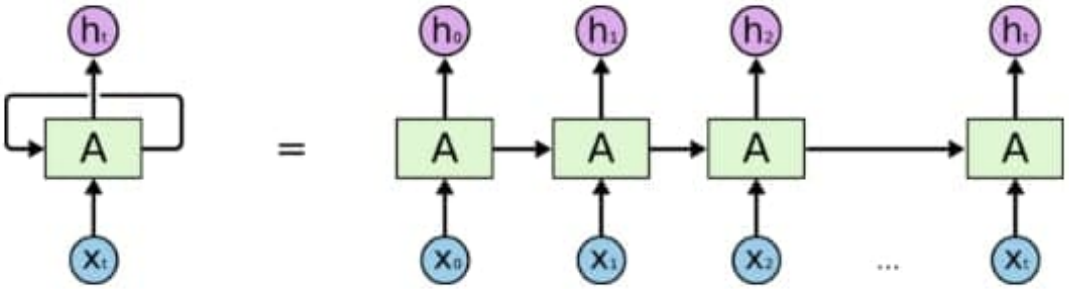
Далее используется функция активации, сжимающая наши значения в интервал [-1;1]:

$y_t = tanh(W_{x_{t-1}} * y_{t-1} - 1 + W_{x_t *} x_t)$, где $W_{x_{t-1}}$,W_{x_t} – матрицы весов предыдущего и текущего входных состояний соответственно.

Таким образом конечный выход рекуррентного слоя можно выразить: $y_t = W_{y_t} * h_t$, где W_{y_t} – матрица весов выходного состояния

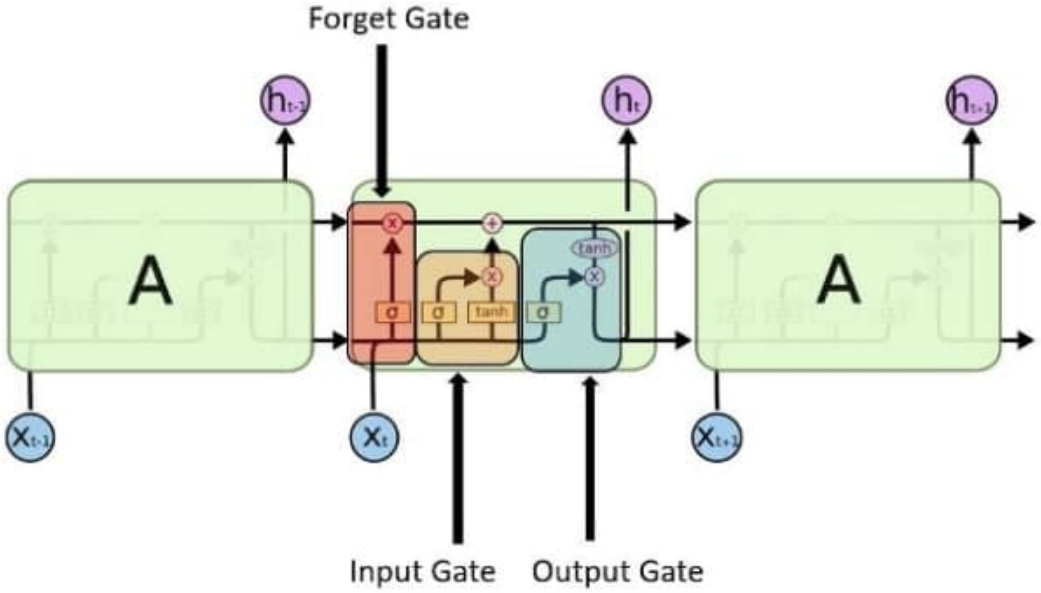
Такие сети применимы в задачах, где нечто целостное разбито на части, так как в отличие от нейросетей прямого распространения, в рекуррентных нейросетях вдобавок к новым данным, которые получает нейрон, передается информация о предыдущем состоянии сети. Однако такие сети имеют проблему : они хорошо помнят недавние данные, но плохо помнят то, что подавалось на вход давно.

Рис. 1: Изображение рекуррентной архитектуры источник[7]



В связи с этим была создана модифицированная RNN[1] сеть LSTM[1] (Long short-term memory). В данной сети изменены ячейки с нейронами: В них добавлен фильтр, отвечающий за то, стоит ли хранить данную информацию или нет. По мере обучения, состояние ячеек будет изменяться: информация добавляться или удаляться из состояния ячейки фильтрами, которые контролируют поток информации на входах и на выходах модуля на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения.

Рис. 2: Изображение рекуррентной LSTM архитектуры источник[7]



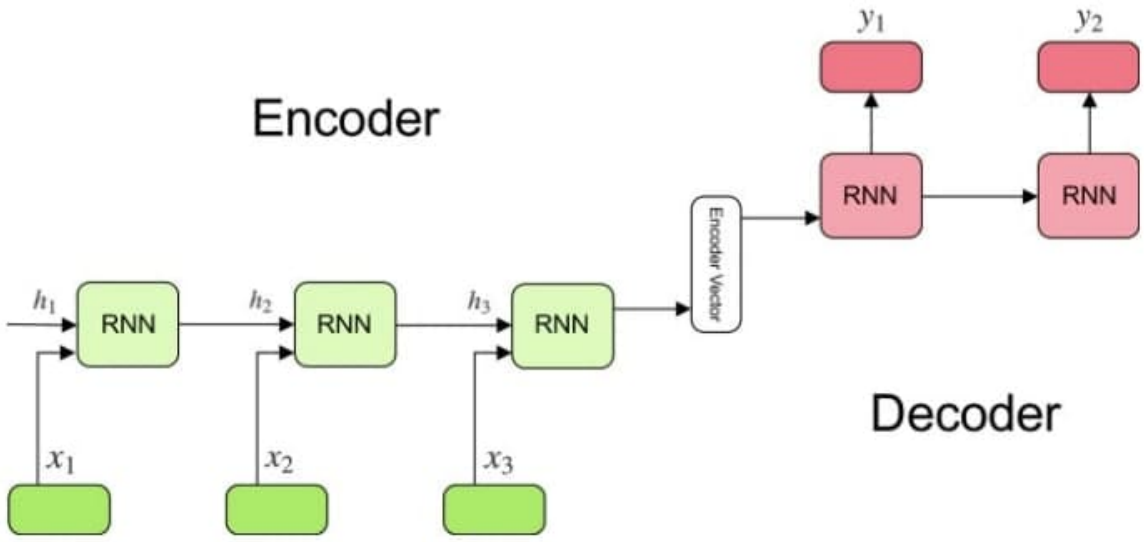
Применяемый фильтр выглядит следующим образом:
$f_t = \sigma(W f_{y_{t-1}x_t} + W f_{x_t y_t})$, Wf – веса сигмоидального слоя. Полученные

значения будут находиться в промежутке [0,1]. Применяться данный фильтр будет как к информации, находящейся на данном шаге, так и к входной и выходной информации, чтобы получить веса ее ”выхода” и ”получения”.

1.2. Seq2seq архитектура

Следующим шагом для решения множества задач с различными данными была создана модель seq2seq (sequence to sequence). Она состоит из двух реккурентных сетей, называющихся encoder и decoder частями. Encoder (кодировщик) отвечает за обработку входных данных и состоит из стека LSTM ячеек, которые отвечают за инкапсуляцию всех входных данных, а также выполняет функцию начального состояния decoder стека. Decoder в свою очередь также состоит из стека LSTM ячеек, каждая из которых принимает предыдущие скрытые состояния и генерирует свое, таким образом выход будет содержать все предыдущие состояния. Выходное состояние определяется при помощи softmax функции для создания вектора вероятности, который поможет нам определить окончательный результат .

Рис. 3: Изображение seq2seq архитектуры источник[5]



Данная архитектура помогает отображать последовательности друг другу, однако она не способна запоминать длинные предложения в связи с большим количеством рекуррентных модулей.

1.3. Code2seq архитектура

В приложении[2] авторы рассказывают нам об альтернативной seq2seq архитектуре, примененной непосредственно к задачам, связанным с представлением кода. Она необходима для его векторного представления.

Ключевым в данной модели является понятие AST — абстрактного синтаксического дерева. Это конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы. Листья такого дерева в дальнейшем будут называться терминалами, а не листья — нетерминалами.

Первоначально строится классическое AST кода, далее предлагается выбрать в нем k случайных путей

Имеется множество $x_1 ... x_k$, $x_i = v_1^i ... v_n^i$ путей в AST дереве. Каждый путь мы представляем при помощи его терминалов, к которым применяем обученные матрицы эмбеддингов $E_{v1}^{nodes} ... E_{vn}^{nodes}$. Получившуюся последовательность кодируем при помощи LSTM блоков encoder части. Таким образом получившийся случайный путь будет представлен следующим образом encode-path$(v_1^i ... v_n^i) = [h_l; h_1]$,где $[h_l; h_1]$ говорит о двунаправленности представленной модели.

Для учета начала и конца пути необходимо добавить начальный и последний терминалы, для их представления обучаются специализированные матрицы эмбеддингов всех токенов, поэтому для представления терминала необходимо просуммировать те матрицы подтокенов, которые входят в состав нашего терминала.

Для получения конечного представления случайного пути применяется tahn-слой к сконкатенированным представлениям пути и ее начального и конечного терминалов.Таким образом на выходе мы получаем комбинированное представление.

В качестве начального состояния декодера используется усреднен-

ный вес представлений всех случайных путей в данном AST дереве

Модель использует стандартную энкодер-декодер LSTM архитектуру с тем отличием, что энкодер не принимает последовательность непрерывно, а каждый AST путь подается отдельно.

Как видно из описания модели, в первоначальной идее мы не можем использовать ее для автодополнения кода, так как она предоставляет нам лишь инструмент для представления кода в виде последовательности.

1.4. Transformer архитектура

В связи с большим количеством рекуррентных модулей и отсутствием способности запоминать много информации, появился Transformer.

Transformer тоже является разновидностью рекуррентной энкодер-декодер архитектуры, одновременно с этим в нем появляется специальный слой, благодаря которому можно отказаться от передачи закодированных скрытых состояний по порядку. Это облегчает распараллеливание сети.

Каждый блок кодировщика состоит из двух слоев: self-attention layer, и обыкновенный слой прямого распространения, между которыми всегда проводится процедура нормализации. На первый слой поступают матрицы эмбеддингов, которые в процессе обучения умножаются на матрицы весов запроса, ключа и значения. Данные матрицы изначально генерируются случайно. Пользуясь ими мы получим формулу для выходного значения матрицы весов Z:

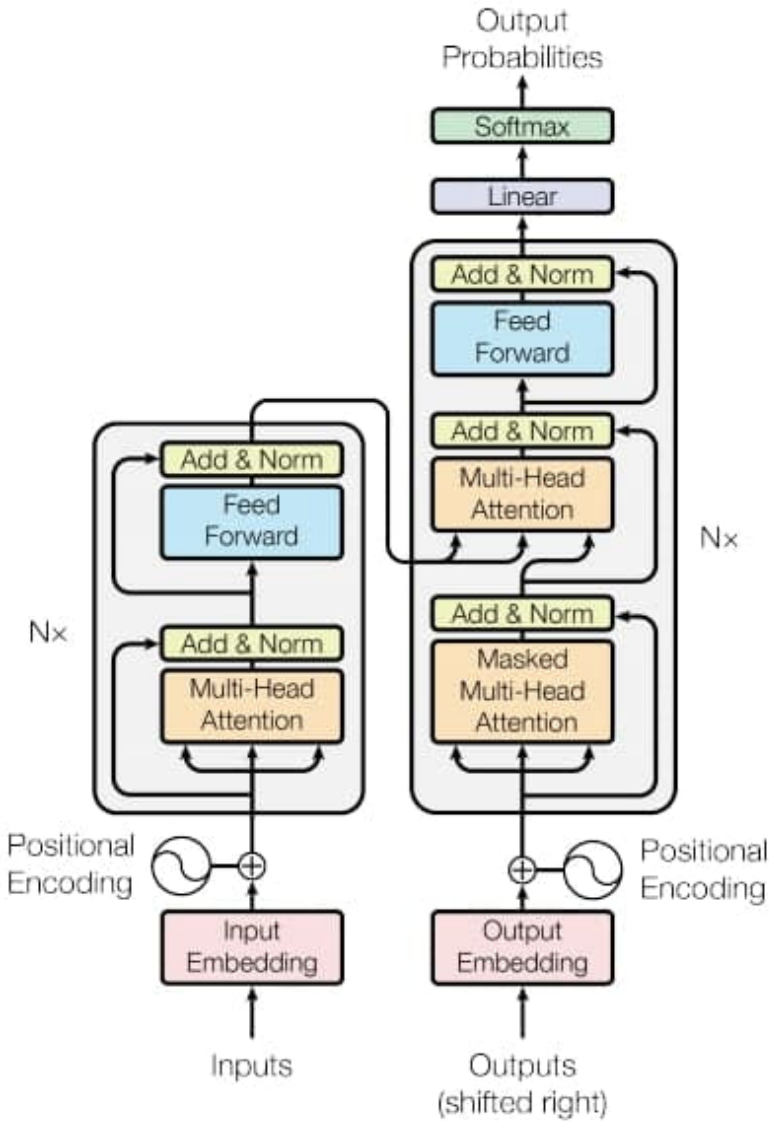
$Attention(Q, K, V) = softmax(\frac{Q*K^T}{sqrt(d_k)} * V)$

Где:

- Q — это матрица эмбеддингов, умноженная на матрицу весов запроса
- K — матрица эмбеддингов, умноженная на матрицу весов ключа
- V — матрица эмбеддингов, умноженная на матрицу весов значения
- d_k — размерность векторов, из которых состоит матрица весов ключа,
- $softmax$ — функция активация, считающая вероятность принадлежности к какому-либо классу

В transformer использован более усовершенствованный, чем вышеописанный, подход, называющийся multi-attention. Ключевое отличие в том, что на каждом слое вычисляется n наборов матриц весов.В то же

Рис. 4: Изображение transformer архитектуры источник[9]



время на слой прямого распространения идет один результат, чтобы получить который генерируется и обучается в процессе дополнительная матрица весов W0, на которую умножаются сконкатенированные матрицы ответов для каждого из наборов. Такой подход помогает сконцентрировать внимание на разных контекстах предсказываемого токена, а каждый из n наборов называется головой.

Также интересным вопросом обучения трансформер архитектуры является существование исследований, показывающих,что полноценный multi-attention механизм не всегда нужен для использования, так как выходы голов зачастую соответствуют некоторым наперед задан-

ным шаблонам. Использовав этот факт, можно сократить количество параметров обучения, а следовательно мощности и время.

Все вышеописанное не решает проблемы порядка слов, который может иметь дополнительный важный контекст. Это тоже учтено в transformer архитектуре, поэтому кодировщику на вход поступает векторизованная последовательность с позиционной информацией. В декодер части помимо вышеописанных self-attention и feed forward слоев присутствует encoder-decoder attention слой. Выход последнего блока кодировщика преобразуется в набор векторов ключа,запроса и значения. Эти вектора используются в каждом encoder-decoder слое кодировщика для сосредоточения на контексте. Декодер также имеет вход, отвечающие за позиционное кодирование.

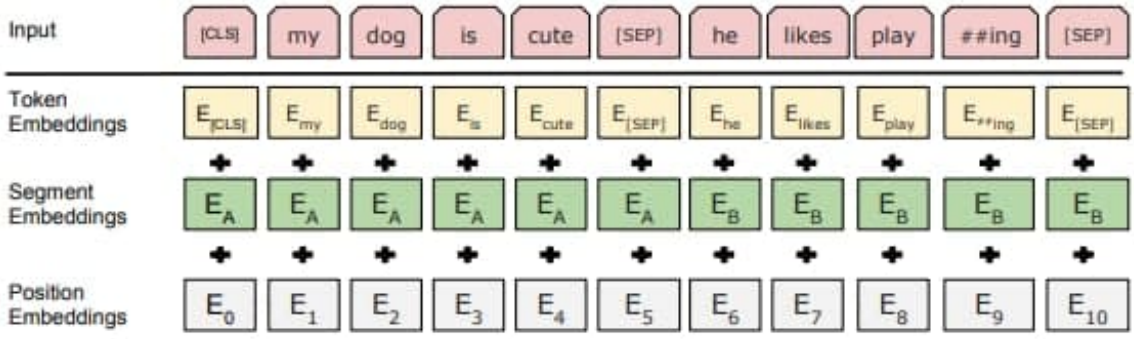
1.5. BERT архитектура

Рассмотрим модель BERT[4]: нейронную сеть от google, которая показала state-of-the-art результаты для многих задач в 2018 году. На данный момент BERT обучена на ряде различных текстовых корпусов, в том числе она была обучена и на корпусе кода.

Особенность архитектуры BERT модели — это усовершенствование архитектуры encoder части модели transformer[9]. Модель предобучена на двух задачах, что позволило ей улучшить свою точность относительно других авторегрессионных моделей.

Первая задача заключается в предсказывании замаскированных токенов. Заранее заданный процент входных токенов маскируется при помощи специальных [mask] символов. В стандартной модели маскируется 15 процентов входных символов, позиции которых выбираются случайным образом. Далее задача обучения состоит в предсказывании замаскированных токенов. Архитектура сети остается неизменной. Авторы называют данную процедуру "Masked LM". В связи с этим эмбеддинги, получившиеся на последнем скрытом слое подаются на функцию активации softmax, определяющую наиболее подходящий токен из существующего словаря.

Рис. 5: Представление входных эмбеддингов модели BERT для задачи предсказания предложения. Они получаются путем суммы эмбеддингов токена, предложения и позиции слова. источник[4]



Во время дообучения [mask]-символы поставить невозможно. Модель смягчает данную ситуацию тем, что лишь 80 процентов выбранных токенов маскируются специальными символами. В данном случае

стоит задача уменьшения перекрестной энтропии. Благодаря двунаправленной организации процесса обучения модель учится учитывать контекст предложения для предсказывания замаскированных токенов. В данной задаче ключевое отличие от трансформера заключается в ее выходе, который по сути представляет собой выход для задачи классификации.

Во втором случае модель учится бинарной задаче предсказания нового предложения по предшествующему. Это необходимо для того, чтобы нейросеть научилась понимать соотношения между разными предложениями. Несмотря на кажущуюся простоту данной задачи, она представляла большую пользу в обучении модели.

1.6. BISON архитектура

Существуют различные способы выбора замаскированных токенов, однако остается нерешенным один из главных недостатков модели BERT. Он заключается в том, что замаскированные токены в этой модели не оказывают влияния друг на друга. В то же время каждый токен может заключать в себе важную информацию. В связи с этим недостатком теряется часть информации и связей между токенами и контекстом. Также в связи со своим двунаправленным подходом в задаче маскирования, мы не можем использовать ее напрямую для генерирования и автодополнения кода, так как в тех условиях полную последовательность необходимо знать заранее. Данные проблемы решает модель BISon[6]. Она адаптирует BERT модель,маскирующую символы, для задачи генерирования текста, учитывая при этом взаимосвязь замаскированных символов друг с другом.

Bidirectional Sequence Generation (BiSon) — сеть, в основе которой лежит двунаправленная авторегрессионная архитектура Transformer.

При генерировании текста , используя подход трансформера, мы предсказываем выходную последовательность y , опираясь только на входную последовательность x. BiSon предлагает генерировать k токен последовательности s, полученной конкатенацией 1...k-1 токена, которые образуют последовательность x, и токенов k+1...n , которые образуют последовательность y,подавая их на вход кодировщику. Используя такой подход, мы имеем больше данных, которые помогут нам точнее узнавать значение k-го токена.

В то же время мы не можем знать будущие токены, поэтому во время предсказывания мы заменяем их специальными замаскированными символами, которые открываем в процессе обучения.

Данная архитектура во время обучения имеет две глобальных подзадачи. Первая заключается в неизвестности последовательности, в которой необходимо открывать замаскированные токены для получения оптимального результата. Вторая задает вопрос, какую стратегию для замены символов на замаскированные стоит выбрать.

Выбирать количество нужно экспериментально при помощи распределения Бернулли или распределения Гаусса. При выборе распределения Бернулли с заданным наперед математическим ожиданием мы выбираем , будет ли этот токен замаскирован или нет, и так для каждого токена в последовательности y.

При выборе распределения Гаусса, количество замаскированных символов высчитывается при помощи нормального распределения с двумя гиперпараметрами (математическое ожидание и дисперсия), а места выбираются случайным образом. Выбирать наиболее подходящий подход необходимо опытным путем.

Проблема открытия [mask]-символов также имеет несколько решений. Наиболее подходящее из которых нужно выбирать опытным путем.

Стратегии открытия замаскированных токенов::

- One step greedy — Все замаскированные токены открываются одновременно, выбирается тот, у которого вероятность появления наибольшая.
- Max probability — Итеративно заменяются токены с наибольшей вероятностью.
- Min entropy — Итеративно заменяются токены с наименьшей энтропией (наименьшей неопределенностью выбора, наименьшим контекстом)
- Left to right —Токены открываются последовательно слева направо
- Right to left —Токены открываются последовательно справа налево
- No look ahead — Слева направо, но не учитываем последующие замаскированные токены. Если такой подход работает хуже, чем просто открытие слева-направо, то это значит, что в последующих замаскированных токенах скрыта важная информация

Авторы использовали данную архитектуру для генерации текстов, но не для генерации кода, в связи с этим наилучшии стратегии выбора замаскированных токенов и способов их открытия для нашей задачи неизвестны, однако для задачи авторов таковыми были распределение Бернулли и открытие токенов слева-направо. Однако для каждой задачи лучше исследовать все вышеописанные способы и выбрать опытным путем самый точный.

2. Эксперимент

2.1. Получение и предобработка данных

Необходимо иметь много данных для обучения модели. Одним из вариантов получения данных является выгрузка существующего кода с открытых баз. Существует база различных проектов github. Они написаны на различных языках, в том числе и на python. Основываясь на этой базе, был создан набор данных 150k python dataset, содержащий адреса репозиториев, содержащих python код.

Часто перед тем, как подать на вход модели данные, как уже говорилось, их необходимо обработать. Существует множество способов сделать это, например решение sentencepiece от компании Google. Это токенайзер текста, который выполняет лексический анализ данных, пытаясь предсказать вероятность появления токена, после этого создает словарь с наиболее используемыми токенам и их численными значениями, которые в дальнейшем используются в качестве словаря.

Также одним из частоиспользуемых, но немного устаревшим, подходом является архитектура word2vec, которая позволяет получить эмбеддинги и векторные представления слов для дальнейшего использования.

Используемые в данной работе сети уже решают проблему предобработки текстовых данных, поэтому представлять заренее представлять их в виде эмбеддингов и составлять словари из самых используемых токенов не понадобится. Однако в связи с нижевозникающей проблемой недостатка вычислительных мощностей, количество используемых репозиториев было ограничено числом 10.

Для обучения BiSon архитектуры необходимо было провести предобработку данных. Она заключалась в вставке символов-разделителей известной и предсказываемой последовательностей. В данном случае символы были поставлены в середину каждой подаваемой последовательности.

2.2. Описание Алгоритма

На основе полученной теории можно выработать несколько алгоритмов. Рассмотрим их:

Одной из идей является слияние Code2seq и BiSon архитектур для достижения наилучших результатов. Изначально используется code2seq архитектура и строится дерево для известной последовательности данных, которую необходимо дополнить. К полученным векторным представлениям случайных путей далее предлагается применить BiSon архитектуру, которая поможет в генерировании последовательностей с учетом контекста. Как конечный результат мы будем использовать ту последовательность, вероятность которой была наибольшей.

Эту идею можно немного облегчить, заменив достаточно сложную BiSon архитектуру моделью Transformer. Это позволит генерировать код, в то же время используя идеи представления кода в виде абстрактных синтаксических деревьев.

Альтернативным способом, который использует сразу обе архитектуры , но и задействует больше ресурсов для обучения является встраивание замаскированных символов (терминалов) в обучение code2seq модели для улучшения векторных представлений кода. Для генерирования используется также BISon архитектура. Как конечный результат мы будем использовать ту последовательность, вероятность которой была наибольшей.

Однако вышеописанные алгоритмы тяжелы в обучении. В связи со сложностью своих архитектур, они требуют большого количества мощностей.

Было проведено исследование, которое показало, что доступных вычислительных мощностей недостаточно для обучения какого-либо из представленных выше алгоритмов. В связи с этим получить результаты по этим алгоритмам не удалось.

Недостаток вычислительных мощностей привел к появлению следующих решений:

Архитектура BiSon не применялась ранее к задаче автодополне-

ния кода, поэтому можно рассмотреть ее использование в поставленных условиях. Для этого необходимо обучить ее на рассматриваемых данных и посмотреть на результаты ее использования. Однако эта нейросеть тоже имеет достаточно сложную архитектуру, в связи с чем обучить ее в полной мере при имеющихся возможностях не получится.

2.3. Результаты

Была выбрана модель BiSon, предлагаемая авторами статьи. Для ее обучения использована среда google colaboratory, предоставляющая сервер с поддержкой GPU емкостью до 13гб видеопамяти. Также лаборатория отличается ограниченным сеансом с максимальной длительностью достигающей 6 часов.

Как сказано в главе, посвященной обработке данных, для обучения BiSon архитектуры необходимо было провести предобработку данных. Она заключалась в вставке символов-разделителей известной и предсказываемой последовательностей. В данной работе символы были поставлены в середину каждой подаваемой последовательности. BiSon основывается на предобученной модели BERT, находящейся в открытом доступе.

В таблице 1 приведены неизменяемые параметры BiSon модели, которые получены при помощи загрузки предобученной модели bert-base-uncased:

Таблица 1: Параметры наследуемые от BERT модели

Значение параметра	**Параметр**
Вероятность в слое dropout для слоя внимания	0.1
Скрытая функция активации	gelu
Вероятность dropout на скрытом слое	0.1
Размерность скрытых состояний	768
Стандартное отклонение для инициализации всех матриц весов	0.02
Размерность слоя прямого распространения	3072
Максимальная размерность эмбеддинга	512
Количество голов в энкодере	12
Количество скрытых слоев в энкодере	12
Размер словаря типа токенов	2
Размер словаря токенов	30522

Стандартные настраиваемые параметры BiSon модели имеют вид,

представленный в таблице 2:

Таблица 2: Стандартные параметры

Значение параметра	**Параметр**
Максимальная длина последовательности	120
Bert-модель	bert-base-uncased
Количество эпох	3
Размер батчей	16
Максимальная длина сгенерированной последовательности	50

Максимальная длина полной последовательности для обучения была уменьшена с 384 сначала до 120 символов, а в дальнейшем до 75.

При обучении модели со стандартными параметрами были получены результаты, отображаемые в таблицах 3 и 4:

Внимательно изучив таблицу, можно сделать вывод, что уже со стандартными параметрами, нейросеть научилась определять,где написана уже завершенная строка; где функция, требующая присутствия переменных; где необходим импорт библиотеки. В то же время качество самого автодополнения нельзя назвать удовлетворительным, так как модель предлагает слишком длинные и перенасыщенные куски кода.

Размышления на эту тему приводят к выводу, что генерируемая последовательность слишком длинная для этой задачи и данных. Также здесь не используется какая-либо стратегия для выбора замаскированных токенов. Для обычного дополнения текстов оптимальном было использование распределения Бернулли в данной задаче, было решено использовать его также для задачи автодополнения. В связи с этим была обучена та же модель, но с другими параметрами, приведенными в таблице 5. Модель была обучена с функцией ошибки равной 6.005.

Основные результаты, полученные при помощи модели, использующей распределение Бернулли приведены в таблице 6. Исходя из результатов, нейросеть уже дополняет код намного более логично и уместно.

Таблица 3: Результаты со стандартными параметрами 1

Код для дополнения	**One step greedy**	**left to right**
import from	torchgp	torche as a torche - based model and a torche - based model as a torche model . a torche model is not required for the following : torche model : a torche model with no dummy model , no dummy model
stripped, prompt = p.rstrip(PROMPT)	p .	(p - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s - s
prompt = torch.tensor(	torch) . , , , , ,	torch . torch + torch . torch + torch . torch + torch . torch + torch . torch + torch . torch . torch . torch . torch . torch

Остается неизученной стратегия, использующая распределение Гаусса для выбора токенов для маскирования. В таблице 7 приведены параметры, используемые для обучения BiSon с распределением Гаусса, ошибка обучения равна 0.55:

Результаты, полученные при помощи модели, использующей распределение Гаусса приведены в таблице 8.

Сравнивая таблицы дополнения для распределений Гаусса и Бернулли, имеющие похожие примеры, можно сделать вывод, что применение стратегий заменения символов на [mask]-символы обоснованы и дают достаточно хорошие результаты даже при обучении на одной эпохе. Показано, что они умеют распознавать импорты, функции и условия.

Таблица 4: Результаты со стандартными параметрами 2

Код для дополнения	**max probability**	**min entropy**
import from	torch path .	opentext archive .
stripped, prompt = p.rstrip(PROMPT)	- p - rs .	- - p .
prompt = torch.tensor(	torch torch torch torch torch torch torch torch torch torch torch torch torch) torch torch torch torch)	torch) / torch . torch (torch torch torch torch torch torch torch torch torch (torch)

Таблица 5: Параметры BiSon модели с использованием стратегии Бернулли

Значение параметра	**Параметр**
Максимальная длина последовательности	75
Bert-модель	bert-base-uncased
Количество эпох	1
Размер батчей	16
Максимальная длина сгенерированной последовательности	10
Стратегия маскирования	Бернулли
Математическое ожидание в схеме Бернулли	0.25

Подход в генерировании кода справа налево не оправдывает себя, исходя из результатов. Это связано с тем, что код логически выстраивается слева направо, благодаря своим операторам и функциям, поэтому предсказать его, используя обратный ход, достаточно сложно. К тому же в задаче автодополнения кода происходит обучение дополнению кода по известной части введенного кода (так называемой левой

Таблица 6: Результаты для BiSon с распределением Бернулли

Код для дополнения	**one step greedy**	**left to right**	**max probability**
import	torch	torch	torch
import numpy as	np np	np	np .
def get(self,	sample))	sample) :	sample,self .
from torch import	tensor	tensor	tensor
if	(	(	(

Таблица 7: Параметры BiSon модели с использованием стратегии Гаусса

Значение параметра	**Параметр**
Максимальная длина последовательности	100
Bert-модель	bert-base-uncased
Количество эпох	1
Размер батчей	64
Максимальная длина сгенерированной последовательности	25
Стратегия маскирования	Гаусса
Математическое ожидание в распределении Гаусса	0.25
Стандартное отклонение в распределении Гаусса	0.1

Таблица 8: Результаты для BiSon с распределением Гаусса

Код для дополнения	**one step greedy**	**left to right**	**max probability**	**min entropy**
import	torch	torch	torch	torch
if	none	none:	self	self.cfg
if temp is	none:	none:	none:	none:
def get(self	)	):	,)	):
With open(	)	):	self.path)	= true)

части, а не правой), поэтому попытка предсказать код справа-налево не заканчивается успехом.

На основе результатов последней обученной нейросети для распределения Гаусса рассмотрим обоснованность применения усложненной архитектуры BiSon, а не архитектуры BERT. При прочих равных условиях рассмотрим предсказывание токенов, не учитывая их влияние друг на друга. Результаты приведены в таблице 9.

Таблица 9: Результаты для BiSon с распределением Гаусса без влияния замаскированных токенов друг на друга

Код для дополнения	**no look ahead**
import	torchtext
if	. . . . .
if temp is	none:
def get(self	):
With open(	):

Результаты предсказывания токенов слева направо, без учета последующих замаскированных токенов,показывают, что такой подход работает хуже, чем просто открытие слева-направо. Это значит, что в последующих замаскированных токенах была скрыта важная информация. Такая информация позволяет сделать нам вывод об обоснованности использования BiSon архитектуры.

Выводы

Изучив современные технологии и их реализации для предсказывания текста, можно выбрать из всего многообразия на настоящий момент ту, которая покажет наиболее удачный результат при заданных условиях. В данной работе рассказаны основные подходы, используемые для решения поставленной задачи, а также исследована ранее не используемая в сфере автодополнения кода архитектура.

Исследования показали, что технология BiSon работает достаточно хорошо, однако для должного ее обучения необходимо большее количество вычислительных мощностей, которые позволят использовать объемные корпуса кода и большее количество эпох для обучения. В таком случае автодополнение будет выглядеть более логичным и правильным. Выводы об обоснованности расширения нейросети на полноценный корпус кода уже можно сделать благодаря результатам, полученным на ограниченных данных. Они показывают себя с достаточно хорошей стороны. Во всех примерах нейросеть может определить, когда ей нужно дополнить функцию, условие или обычную строку. Также учитывание влияния замаскированных токенов друг на друга выгодно отличалась на фоне своего контрпримера.

Наиболее успешными параметрами себя показали параметры выбора распределения Бернулли и Гаусса и практически все схемы открытия замаскированных токенов, не включая схему открытия справа налево и слева направо без учета последующих [mask]-символов.

Для небольшого датасета может быть достаточно и обучения в одну эпоху в целях предупреждения переобучения. В сравнении с оригинальной статьей, где BiSon применен к корпусу обычного текста,стандартные параметры, используемые авторами, расходятся с оптимальными параметрами, полученными в результате эксперимента.

Эксперимент показал, что показана возможность переноса методов генерации текста из NLP задач для задачи автодополнения кода.

Заключение

Было изучено множество различных архитектур для решения поставленной задачи. Выбранная для исследования архитектура BiSon содержит в себе важные идеи использования замаскированных токенов для выявления дополнительного контекста между ними, улучшающего результаты генерирования последовательностей.

Интересной также является идея векторного представления кода в виде деревьев, а в дальнейшем и в виде последовательностей, и ее использование совместно с другими архитектурами для автодополнения кода. Были получены теоретические алгоритмы, позволяющие это сделать. В результате проведенного эксперимента были получены положительные результаты, которые могут являться индикатором оправданности использования данной архитектуры.

Проведенные исследования показали, что методы и нейросети, применяемые для задач обработки естественного языка, также можно успешно перенести и расширить на класс задач, связанный с автодополнением кода.

Список литературы

[1] A Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. — Access mode: https://arxiv.org/abs/1808.03314 (online; accessed: 09.08.2018).

[2] Alammar Jay. Illustrated transformer. — 2018. — Access mode: http://jalammar.github.io/illustrated-transformer/ (online; accessed: 27.06.2018).

[3] Alon U. Levy O. Brody S. Yahav F. code2seq: Generating Sequences from Structured Representations of Code. — Access mode: https://arxiv.org/pdf/1808.01400.pdf (online; accessed: 04.08.2018).

[4] Devlin J. Ming-Wei Chang M. Lee K. Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. — Access mode: https://arxiv.org/abs/1810.04805 (online; accessed: 11.10.2018).

[5] Kostadinov Simeon. Understanding encoder decoder sequence-to-sequence model. — 2019. — Access mode: https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04a (online; accessed: 05.02.2019).

[6] Lawrence C. Kotnis B. Niepert M. Attending to Future Tokens For Bidirectional Sequence Generation. — Access mode: https://arxiv.org/abs/1908.05915 (online; accessed: 16.08.2018).

[7] Mittal Aditi. Understanding encoder decoder sequence-to-sequence model. — 2019. — Access mode: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e (online; accessed: 12.06.2019).

[8] Sutskever I. Vinyals O. Le Q.V. Sequence to Sequence Learning with Neural- Networks. — Access mode: https://arxiv.org/abs/1409.3215 (online; accessed: 10.09.2014).

[9] Vaswani A. Shazeer N. Parmar N. Uszkoreit J. Jones L. Gomez A. Kaiser L. Polosukhin I. Attention Is All You Need. — Access mode: https://arxiv.org/abs/1706.03762 (online; accessed: 12.06.2017).